VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

LCD textový terminál

LCD text terminal

2014

Patrik Slučiak

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Patrik Slučiak**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: LCD textový terminál
LCD Text Terminal

Zásady pro vypracování:

Navrhněte připojení LCD víceřádkového displeje jako terminálu pro Unixové systémy. Realizaci proveďte pomocí mikrokontroléru s připojením přes USB rozhraní.

1. Seznamte se s fungováním textových terminálů a jejich standardy.
2. Vyberte vhodný mikroprocesor s USB rozhraním pro připojení LCD displeje.
3. Navrhněte prototypové zapojení zařízení.
4. Realizujte potřebné programové rozhraní mikropočítače.
5. Zhodnoťe stabilitu a rychlost navrženého řešení a jeho integraci do OS Linux.

Seznam doporučené odborné literatury:

[1] W. Richard Stevens, Stephen A. Rago: Advanced Programming in the UNIX Environment, ISBN-10: 0201433079
[2] Datové listy k vybranému procesoru a LCD

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Olivka**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prehlasujem, že som túto bakalársku prácu vypracovala samostatne. Uviedla som všetky literárne parametre a publikácie, z ktorých som čerpala.

V Ostrave 7.mája 2014 .....

Rád by som poďakoval vedúcemu bakalárskej práce, pánovi Ing. Petrovi Olivkovi za metodickú a odbornú pomoc pri spracovaní tejto práce.

Abstrakt

V teoretickej časti tejto bakalárskej práce sú rozobrané štandardy terminálov, spôsob komunikácie terminálov, popis rozhraní (RS232, RS485, RS422), ich princípy a spôsob prenosu dát. Je tu podrobný rozpis USB zbernice, princíp rozoznávania zariadení a prenosu dát. Ďalej sú to popísané konfigurácie procesoru a práca LCD modulov - jednotlivé režimy, oživovanie, zápis. Sú tu uvedené vlastnosti jednotlivých modulov, ktoré je potrebné poznať pri návrhu a vývoji LCD Textového Terminálu komunikujúceho prostredníctvom USB zbernice. V praktickej časti bakalárskej práce, sú zrealizované možnosti návrhu, tak ako aj samotný návrh zariadenia.

Klíčová slova: textové terminály, LCD, USB, CDC, VT, Ascii, CDC, ACM

Abstract

In the theoretical part of this master thesis are descripted terminals standards, terminal communication methods, interfaces descriptions (RS232, RS485, RS422), their principles and the way of data transmission. There are detail description of USB bus, principles of device recognition and data transmission on the bus. In next part of thesis are described MCU configuration and work of LCD modules - individual modes, initialisation, writing characters on the LCD. There are stated properties of modules, which are needed to know for development of LCD text terminal which comunicates via the USB bus. In the practical part of this master thesis are realized options of device design, and also the independent device development.

Keywords: text terminal, LCD, USB, CDC, VT, Ascii, CDC, ACM

Seznam použitých zkratek a symbolů

ROM	–	Read Only Memory
MCU	–	Micro Controller Unit
ASCII	–	American Standard Code
SW	–	Software
USB	–	Universal Serial Bus
LCD	–	Liquid-Crystal Display
VT	–	Virtual Terminal
DIP	–	Dual In-line Package
SPDIP	–	Shrink Plastic Dual In-line Package
SOIC	–	Small Outline Integrated Circuit
SSOP	–	Shrink Small Outline Package
QFN	–	Quad-Flat No-leads
TQFP	–	Thin Quad Flat Package
GND	–	Ground
VDD	–	Voltage
DDRAM	–	Double Data Random Acces Memory
D+	–	Data+
D-	–	Data-
CDC	–	Communications Device Class
CIC	–	Communications Interface Class
ACM	–	Abstract Control Model
RS	–	Recommended Standard
LCM	–	Load Control Module
OTG	–	On-The-Go

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Cieľom tejto bakalárskej práce je zoznámiť sa s fungovaním textových terminálov, ich spôsobu komunikácie, štandardmi a z poznatkov navrhnúť prototypové zariadenie LCD terminál. Zároveň je potrebné navrhnúť softwarové vybavenie pre požadovaný typ procesora. Táto bakalárska práca sa skladá z teoretickej časti a z praktickej časti obsahujúcej samotný návrh zapojenia, programového vybavenia a nasadenia pod systém Unix. Bakalárska práca súvisí s prácou Radima Žižky „Připojení USB klávesnice k mikropočítači“.

2 Popis textových terminálov

Textové terminály sú zariadenia, ktoré poskytujú vzdialené užívateľské rozhrania. Prvé textové terminály vznikli ako jednoduché zariadenia pozostávajúce z dvoch častí :

klávesnice a **monitora**. Používali sa ako súčasť sálových počítačov, ktoré neboli prenosné a nachádzali sa v uzavretých miestnostiach, na ktorých bežali programy. Odoslaný znak z klávesnice sa vyhodnotil ako vstup do programu bežiaceho na sálovom počítači. Ak program znak rozpoznal, odoslal ho naspäť programátorovi do monitora. Komunikácia prebiehala pomocou sériovej linky, prípadne modemu. Textové terminály vznikli ako požiadavka na vzdialenú obsluhu sálových počítačov a používajú sa dodnes. Terminály používajú štandard **VT100** a jeho nadstavby.

Princíp dnešných textových terminálov sa nezmenil. Dnešné textové terminály obsahujú grafický interface, ktorými sú rôzne typy LCD displejov. Súčasťou dnešných terminálov je procesorová jednotka **MCU unit** obsluhujúca rôzne grafické interfaces, tak ako aj spracovanie vstupne/výstupných dát. Oproti minulosti, dnešné terminály dokážu ponúkať riešenie potrieb aj ako samostatná jednotka bez nutnosti implementácie vstupného zariadenia (klávesnice). Takéto terminály sa používajú ako informačné a prípadnú postrádateľnosť klávesnice terminálu nahradzuje klávesnica počítača.

Textové terminály komunikujú prostredníctvom :

- Sériového rozhrania:
  - **RS232**
  - **RS485**
  - **RS422**
- Paralelného rozhrania:
  - Terminály APT130 pre riadiace systémy:
    - **AMiRiS99**
    - **AMAP99**

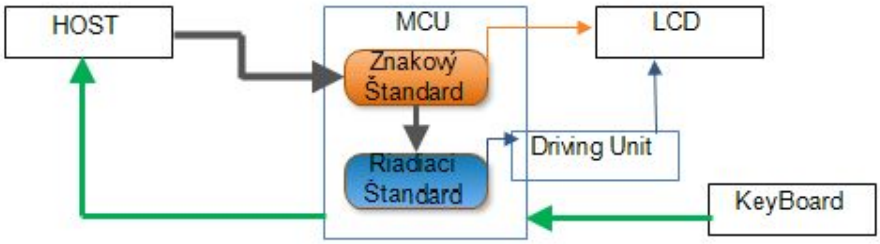
2.1 Princíp činnosti textových terminálov

Textové terminály ako štandardizovaný celok komunikujú cez rozhranie pripojené k zariadeniu v roli master (host) prostredníctvom znakovej sady ANSII.

V našom prípade je hostujúcim zariadením počítač. Znakové sady reprezentujú jednotlivé znaky ako hodnotu v dvojkovej sústave, podľa ASCII tabuľky.

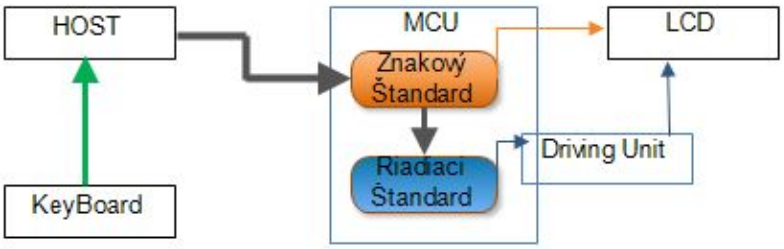
- **Terminály s klávesnicou**

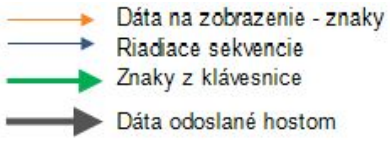
  Procesorová jednotka príjme znaky z klávesnice, odošle ich cez rozhranie hostu (počítači), obslužný program (napr. virtuálny terminál) ich spracuje a odošle naspäť procesorovej jednotke.



- **Terminály bez klávesnice**

  Terminál komunikuje priamo s hostom. Obslužný program mu prostredníctvom rozhrania odosiela dáta taktiež v dvojkovej sústave.





Spracovanie prijatých dát MCU
MCU textového terminálu má svoje štandardy uložené v pamäti.

Po prijatí dát MCU prijaté znaky porovná so svojím štandardom. Pokiaľ sa jedná o riadiace znaky, MCU ich spracuje podľa štandardov **VT100, VT200**,.. a vykoná definovanú funkcionalitu v závislosti od typu terminálu. Terminály sú schopné komunikovať s hostom obojsmerne.

2.2 Komunikácia textových terminálov

Prenos dát HOST $\longleftrightarrow$ Terminál

Najčastejším prenosom je **sériový prenos** realizovaný:

1 **RS-232**
2 **RS-485**
3 **RS-422**

2.2.1 RS-232 rozhranie

Najpoužívanejšie sériové rozhranie textových terminálov.
RS-232 je sériová linka poskytujúca asynchrónny prenos dát. Signály sú reprezentované napäťovými úrovňami voči zemi.
Obsahuje jeden pár vodičov každého smeru IN/OUT, po ktorých sú jednotlivé bity odosielané za sebou.
Prenos dát je full-duplex, terminál s hostom môžu prijímať/odosielať dáta v reálnom čase.
Dáta sú cez RS-232 odosielané do Terminálu v sekvenciách (rámcoch) za sebou. Komunikácia začne prebiehať po vyslaní START bitu host zariadením, jednotlivé rámce sa začnú do terminálu odosielať. Po prenesení všetkých rámcov host odošle STOP bit pre ukončenie spojenia s terminálom.

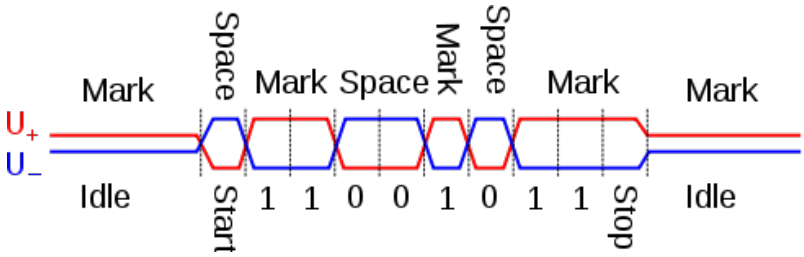
2.2.2 RS-485 rozhranie

RS-485 sa používa ako rozhranie hlavne pre priemyselné terminály.

Pomocou tohto rozhrania môže súčasne komunikovať 32 vysielačov a 32 prijímačov, čo sa využíva pri viacerých terminálov napojených na jednej zbernici naproti RS-232, kde sa komunikácia reprezentuje ako Point-to-Point. Neaktívne hostovké zariadenia , tak ako aj terminály musia byť pri neaktivite v stave vysokej impedancie, aby neovplyvňovali komunikáciu medzi aktívnymi zariadeniami. Viacero terminálov môže poskytovať služby potrieb viacerým hostom, no v reálnom čase môže byť aktívny len jeden host.
Pri vysielaní sa používa diferenciálne kódovanie dát.

Kódovanie dát reprezentujú dve polarity :

Pri prenose host začne komunikáciu START bitom, pričom začne vysielať 8 bitový rámec, za ktorým doplní paritný bit a spojenie ukončí 2x STOP bitom.



Obrázek 1: Princíp prenosu RS-485

2.2.3 RS-422 rozhranie

RS-422 je staršia varianta rozhrania RS-485, ktorá sa taktiež používa v priemyselných terminálov.
Princíp a činnosť RS-422 je ten istý ako u RS-485. Rozdiely medzi oboma rozhraniami sú tie, že rozhranie RS-422 je mupltidrop, na rozhraní je len jedno host zariadenie a maximálne 10 terminálov. Terminály je možné u oboch rozhraniach od hostu umiestniť maximálne 1200m.

2.3 Terminály a USB rozhranie

USB (Univerzal Serial Bus) je typ rozhrania, ktoré bolo vyvinuté za účelom čo najuniverzálnejšej komunikácie medzi zariadeniami.
Ide o sériovú polo duplexnú linku, ktorá podporuje takzvané módy prenosov pre rôzne typy zariadení:

Riadiace

Asynchrónne Bulk, Interrupt

Izochrónne - vzorkovanie a prevedenie na číslicovú formu

Jedným z črtov USB je princíp **Plug-and-Play**. Ide o možnosť pripojenia zariadenia k hostu bez toho, aby sa host musel resetovať. Po pripojení nastane rozpoznanie zariadenia a konfigurácia zariadenia. Spustenie ovládača pripojeného zariadenia a následne komunikácia. Klientske zariadenie je možné v reálnom čase kedykoľvek odpojiť bez výskytu interných chýb.
Na rozdiel od ostatných zariadení, komunikačný protokol USB spočíva v odosielaní paketov dvoch typov :

A **Riadiace pakety**
B **Dátové pakety**

2.3.1 Izochrónny prenos

Izochrónny prenos sa používa pri prenose v reálnom čase - video, zvuk. Pri chybe alebo zlyhaní prenosu sa dáta znovu nepreposielajú, pretože dáta by neboli v reálnom čase aktuálne. Tento druh prenosu (dáta) požadujú určité oneskorenie a konštantnú šírku prenosového spojenia. Pri izochrónnym prenose sa jedná vždy len o jednosmerný prenos dát. Maximálna veľkosť paketu pre full-speed je 1203 bytov. Nízko rýchlostné zariadenia v režime low-speed tento druh prenosu nepodporujú. Koncový bod pri nadväzovaní spojenia, musí definovať oneskorenie - interval prístupu k usb zbernici. Zároveň musí dáta odosielať s nižšou paritou. Pokiaľ koncový bod nemá spracované dáta na odoslanie, odosiela prázdny mikrorámec

2.3.2 Asynchrónny prenos

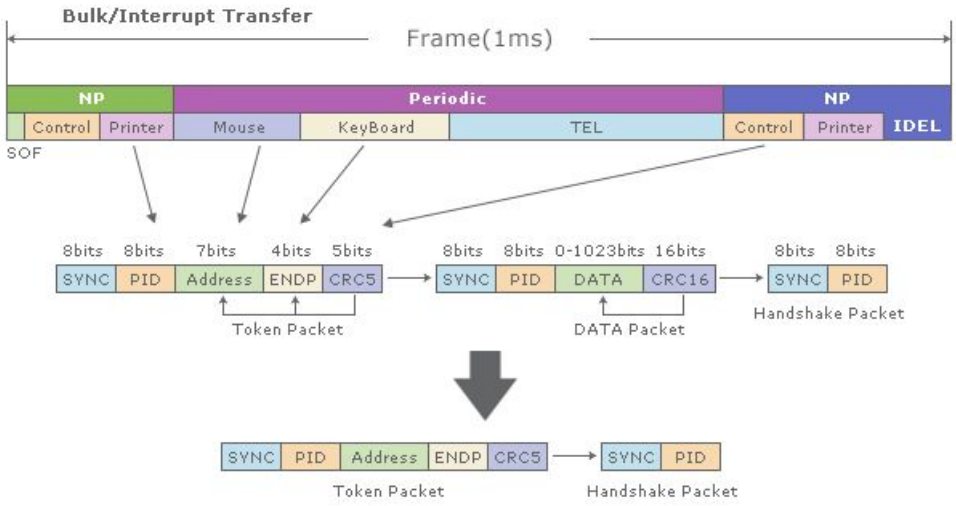
Asynchrónne prenosy, takzvané interrupty, sú nepravidelné a časovo citlivé. Prerušenia ako také vznikajú v koncovom bode alebo v hostovkom zariadení (počítač). Pri vyvolaní udalosti je volaný iterrupt, ktorý čaká na svojom end-pointe, kým si ho prevezme host. Tieto prenosy sú určené pre zariadenia, ktoré nepotrebujú odosielať dáta často, ale vyžadujú znovu odosielanie dát v prípade chyby (myš). Pri tomto prenose sa kladú požiadavky, aby boli dáta prijaté v čo najkratšom čase a boli uprednostnené pred prenosmi bulk a izochrónnymi prenosmi. Maximálna veľkosť dát je pre low-speed 8 bytov, pre full-speed 64 bytov, pre hight-speed 1024 bytov. Pokiaľ koncový bod nemá spracované dáta na odoslanie, odosiela signál "nak".

2.3.3 Prenos Bulk

Bulk prenosy sú určené pre prenos malých, nepravidelne odosielaných dát, pri ktorých sa vyžaduje aby boli prenesené správne. Pokiaľ nastala chyba, dáta sa znovu prepošlú. Ďalšou požiadavkou je kvalita prenosu. Využívajú sa pri USB tlačiarňach. Bulk prenosy sú v poradovníku s najmenšou prioritou a pri komunikácii s USB sa zinicializujú ako posledné. Prenosová rýchlosť je pri full-speed 8-16-32-64 bytov, pri hight-speed maximálne 512 bytov. Zariadenia typu low-speed (nízko rýchlostné) tento druh prenosu nepodporujú.

2.3.4 Riadiace prenosy

Riadiace prenosy (Control Transfer) sú využívané systémovým SW pre konfiguráciu koncového bodu hneď po jeho pripojení. Prenos zaháji host odoslaním transakcie SETUP. Po nej sú do koncového bodu odosielané ďalšie dodatočné dáta. Doručenie dát je kontrolované proti chybám a je bezdrôtové. Po konfigurácii host odošle transakciu STATUS, ktorá ho informuje o stave prevádzanej konfigurácie zariadenia. Pomocou tohto typu prenosu je možné pristupovať k častiam koncového bodu. Taktiež slúži na zistenie informácií o pripojenom zariadení - názov, typ, vnútornú konfiguráciu, podporované end-pointy a podobne. Rýchlosť pre low-speed 8 bytov, pre full-speed 64 bytov, pre high-speed 8,16,32,34 bytov. Pri adresovaní zariadenia, Host odošle riadiace signály, tak ako aj pri potvrdení prijatia dát. Tým je zabezpečená bezchybnosť prenosu dát. Riadiacimi paketmi sa kontrolujú prípadné chyby, ktoré pri prenose nastali.



Obrázek 2: Adresovanie a vytváranie paketov riadiacich(Token)/dátových

2.3.5 USB Device Class

USB ako také ponúka možnosti takzvaných **USB Device Class** [2]. Aby nebolo potrebné ku každému zariadeniu dodávať vlastný ovládač, sú zariadenia rozdelené do jednotlivých tried. Host disponuje jedným ovládačom pre jednu triedu a tudíž jedným ovládačom pre všetky zariadenia v triede. USB Device Class obsahujú podtriedy v ktorých sú zavedené komunikačné protokoly. Pri adresovaní zariadenia systém zvolí taký režim prenosu, aké má dané zariadenie nakonfigurované. Konfigurácia zariadenia je nastavená v jeho descriptore, kapitola 2.3.7

Pri starších typoch procesoroch, ktoré nepodporujú USB rozhranie, sa hardwarovo vytvorí takzvaný **USB - RS-232 bridge** .

Ide o prevodník rozhraní [HOST USB $\Longleftrightarrow$ RS-232 MCU].

USB bridge sa dá riešiť aj prostredníctvom **SW**. Novšie typy MCU Unit, ktoré podporujú USB interface umožňujú implementovať firmware [4], ktorý programovo vytvorí premostenie komunikácie priamo v procesorovej jednotke. Jednou z USB device class je trieda **Communications and CDC Control**, tabuľka 1. Firmaware nastaví MCU Unit do módu **CDC**. Pri adresovaní host CDC zariadenie zavedie do režimu CDC. Zariadenie (terminál) pripojené k hostu ako CDC je pripojené cez USB rozhranie, ale v počítači sa tvári ako virtuálny COM Port, teda ako rozhranie RS-232.

Kód triedy	Typ - protokol	Názov
00h	Zariadenie - 00h	Use class information in the Interface Descriptors
01h	Rozhranie - xxh	Audio
02h	Rozhranie/Zariadenie - xxh	Communications and CDC Control
03h	Rozhranie - xxh	HID (Human Interface Device)
05h	Rozhranie - xxh	Physical Device Class
06h	Rozhranie - 01h	Image
07h	Rozhranie - xxh	Printer
08h	Rozhranie - xxh	Mass Storage
09h	Zariadenie - 00h, 01h, 02h	Hub
0Ah	Rozhranie - xxh	CDC-Dáta
0Bh	Rozhranie - xxh	Smart Card
0Dh	Rozhranie - 00h	Content Security
0Eh	Rozhranie - xxh	Video
0Fh	Interface - xxh	Personal Healthcare
10h	Rozhranie - 00h	Audio/Video Devices
DCh	Rozhranie/Zariadenie - 01h	Diagnostic Device
E0h	Rozhranie - 01h, 02h, 03h	Wireless Controller
FEh	Rozhranie - 01h, 00h	Application Specific
FFh	Rozhranie/Zariadenie - xxh	Vendor Specific

Tabulka 1: USB Device class

2.3.6 Podtriedy triedy CDC (Communications and CDC Control)

Terminály ako zariadenia nakonfigurované do triedy, Communications and CDC Control„ sú rozšírené o **Communications Interface Class - CIC** a využívajú podtriedu **Abstract Control Model - ACM** triedy CDC, tabuľka 2. Ako nadstavbu komunikačného protokolu využíva **V.250** - (kód triedy: **01h**), čo je sada štandardov pre modemy.

Podtrieda ACM implementuje 5 rozhraní [3] pre zabezpečenie komunikácie - SendEncapsulatedCommand, GetEncapsulatedResponse, SetLineCoding, GetLineCoding, SetControlLineState.

SendEncapsulatedCommand - slúži na vyslanie príkazu vo formáte, ktorý podporuje kontrolný protokol triedy communications class - V.250.

GetEncapsulatedResponse - používa sa pri požadovaní odpovedi na formát kontrolného protokolu - triedy comunications class.

SetLineCoding - používa sa na kódovanie dát na linke. Umožňuje hostovi špecifikovať formát vlastnosti znakov. Pri emulácii RS-232 je táto funkcionalita dôležitá. Host a terminál si musia byť vedomé akou rýchlosťou sa dáta prenášajú, aký počet stop bitov bol prenesený a o paritnom bite.

GetLineCoding - umožňuje hostovi zistiť aktuálne nakonfigurovanú linku a prevziať si kódované parametre.

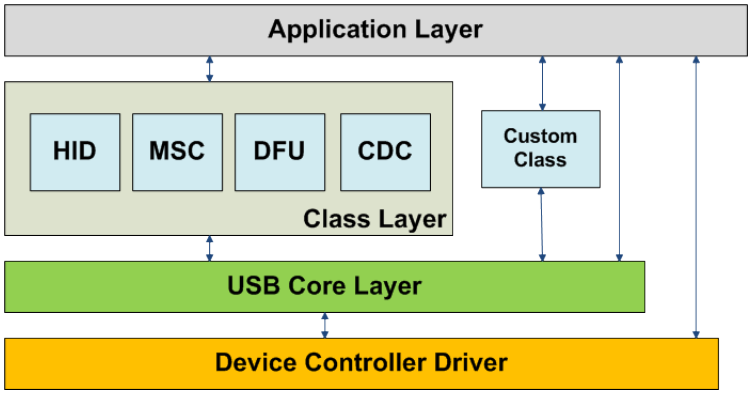
SetControlLineState - generuje riadiaci tok signálov RS-232. Kontrola linky signálu (Control Line Signal) CLS.

SetLineCoding, GetLineCoding - pri kódovaní je použité známkovanie dát. Dáta sa známkujú bitmi za sekundu. Pri prenose sa používajú stop-bity, ktoré sa môžu zabaľovať po: 0 - 1, 1 - 1.5, 2-2. Paritný byt môže nadobúdať hodnoty od 0-4 : 0 = None, 1 = Odd, 2 = Even, 3 = Mark, 4 = Space. Dáta sa môžu odosielať v 5, 6, 7, 8 alebo 16 bitových rámcoch.

Kód triedy	Názov
01h	Direct Line Control Model
02h	Abstract Control Model
03h	Telephone Control Model
04h	Multi-Channel Control Model
05h	CAPI Control Model
06h	Ethernet Networking Control Model
07h	ATM Networking Control Model
08h	Wireless Handset Control Model
09h	Device Management
0Ah	Mobile Direct Line Model
0Bh	OBEX

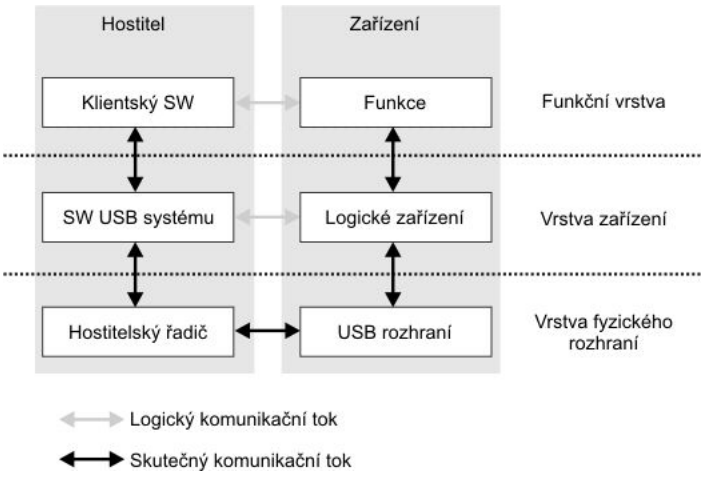
Tabulka 2: Podtriedy triedy CDC

Jednotlivé triedy zariadení USB Device Class sa nachádzajú v triednej vrstve (Class Layer), ktorá je medzi aplikačnou a fyzickou vrstvou procesora. Komunikácia Class Layer je obojsmerná pre obe vrstvy. Fyzická a aplikačná vrstva navzájom komunikujú a sú riadené pomocou firmware MCU. Grafický návrh na obrázku 3.



Obrázek 3: Vnútorná schéma USB Device Class

Implementácia vrstiev pri logickej komunikácií, obrázok 4, sa rozdeľuje na **funkčnú vrstvu**, **vrstvu zariadenia** a **fyzickú vrstvu**. Implementácia na jednotlivých vrstvách sa v hostu a v zariadení líši. Funkčná vrstva je u hosta tvorená klientskym, obslužným softwarom, zatiaľ čo u MCU je tvorená metódami (funkciami). Logický komunikačný tok sa odohráva medzi jednotlivými vrstvami zariadení, pričom logicky komunikujú len rovnaké vrstvy oboch zariadení. Reálny komunikačný tok prebieha len na fyzickej vrstve, cez ktorú sú obe zariadenia prepojené. Device layer je u hosta implementovaná systémovým softwarom (driverom) USB, pri MCU je tvorená logickým zariadením. Na fyzickej vrstve má host zavedený radič, pri MCU je zavedené USB rozhranie. Odosielané dáta hosta sa spracujú od aplikačnej vrstvy, cez vrstvu zariadenia a po fyzickej vrstve sa v podobe binárneho kódu odošlú po zbernici do koncového bodu, v ktorom sa od fyzickej vrstvy cez vrstvu zariadenia spracovávajú až k aplikačnej vrstve.



Obrázek 4: Rozvrstvenie implementácie vrstiev HOST - Terminál

2.3.7 Descriptor zariadenia

Descriptory sú dátové štruktúry v ktorých je konfigurácia zariadenia. Poznáme descriptory rozhrania, zariadenia, koncového bodu, nastavení, ladiace (debug) a podobne. Všeobecne, jeden descriptor rozhrania môže špecifikovať ďalšie descriptory koncových bodov. Descriptory ďalej definujú počet rozhraní a konfiguráciu zbernice. V prípade descriptora zariadenia, po zapojení zariadenia do USB zbernice je host schopný zistiť informácie o zariadení ako názov, verziu, triedu USB Device Class,... Každé zariadenie ma univerzálne PID - identifikačné číslo, podľa ktorého host vie s akým zariadením má komunikovať v prípade, že je na zbernici viacej koncových bodov. Pri komunikácií s hostom, koncový bod odosiela svoj descriptor.

Descriptor zariadenia je v tomto prípade nakonfigurovaný:

```
ROM USB_DEVICE_DESCRIPTOR device_dsc=
{
    0x12,         // velkost  descriptora  (v bytoch)
    USB_DESCRIPTOR_DEVICE, // typ descriptora zariadenia
    0x0200,               // verzia  specifikacie  USB v BCD formate
    CDC_DEVICE,           // kod  triedy
    0x00,                 // kod podtriedy
    0x00,                 // kod pouziteho  protokolu
    USB_EP0_BUFF_SIZE, // maximalny pocet paketov pre koncoy bod  0)
    0x04D8,               // Id  vyrobca
    0x000A,               // Id  Produktu CDC RS-232 Emulation Demo
    0x0100,               // cislo  verzie  prouktu v BCD formate
    0x01,                 // index  stringu  vyrobca
    0x02,                 // index  stringu  produktu
    0x00,                 // index  stringu  serioveho  cisla  zariadenia
    0x01                  // pocet  moznyh  konfiguracii
};
```

Výpis 1: Descriptor zariadenia

2.3.8 Descriptor konfigurácie

Descriptor nastavenia sa ako descriptor zariadenia taktiež nachádza priamo v koncovom bode. Obe konfigurácie sú zavedené v pamäti Flash MCU. Descriptor nastavení rozširuje konfiguráciu ďalších descriptorov a definuje navyše **descriptor rozhraní**, **descriptor špecifikácie CDC triedy** a **descriptor koncového bodu**.

Pri firmware ako pri univerzálnom celku, descriptor nastavenia rozširuje viacero koncových bodov z dôvodu možnosti použitia rôznych typov prenosov.

```
ROM BYTE configDescriptor1[]=
{
    0x09,                          // velkost  descriptora  (v bytoch)
    USB_DESCRIPTOR_CONFIGURATION, // typ descriptora nastavenia
    67,0,                          // celkova  dlzka  dat  pre  descriptor
    2,                             // cislo  rozhrania
    1,                             // Index  hodnoty  tejto  konfiguracie
```

```
    0,                              // Index stringu konfiguracie
    _DEFAULT | _SELF,               // atributy (pri inite zbernice)
    50,                             // maximalna spotreba energie (2X mA)

    9,                              // velkost descriptora rozhrania (v bytoch)
    USB_DESCRIPTOR_INTERFACE, // typ descriptora rozhrania
    0,                              // cislo rozhrania
    0,                              // alternativne nastavenie
    1,                              // pocet koncovych bodov v tomto interface
    COMM_INTF,                      // kod triedy
    ABSTRACT_CONTROL_MODEL, // kod podtriedy
    V25TER,                         // kod pouziteho protokolu
    0,                              // Index stringu rozhrania

    0x07,                           // velkost descriptora koncoveho bodu (v bytoch)
    USB_DESCRIPTOR_ENDPOINT, // typ descriptora nastavenia
    _EP01_IN,                       // adresa koncoveho bodu
    _INTERRUPT,                     // druh prenosu
    0x08,0x00,                      // velkost
    0x02,                           // tnterval prenosu
}
```

Výpis 2: Descriptor nastavenia

2.4 Štandardy terminálov

Štandardy ako také pozostávajú zo sady dohodnutých znakov, ktoré zariadenia používajú pre svoju kompatibilitu s inými zariadeniami. Zavedením štandardov sa vybudovala jednotná komunikácia a funkcionalita pre zariadenia, ktoré štandardy používajú. Štandardy sú reprezentované v podobe binárnej sústavy.

Pokiaľ znak vyhovuje, tak je prepustený na ďalšie spracovanie prípadne sa vykoná implementovaná funkcionalita pridelená práve tomuto znaku ako súčasti štandardu. Pokiaľ znak nevyhovuje danej tabuľke štandardu, zahodí sa.

Podľa druhu terminálu a toho akú výstupú zobrazovaciu jednotku implementuje sa odvíja štandard, ktorý musí pre dané zobrazenie na LCD a funkčnosť zodpovedať. Textové terminály majú implementovaných väčšinou viacero štandardov.

Znakový štandard

ASCII

Riadiace štandardy VT

VT05
VT50, VT52, VT55, VT62
VT100, VT101, VT102, VT103, VT105, VT110, VT125, VT131, VT132
VT200, VT220, VT240, VT241
VT300, VT320, VT330, VT340
VT420
VT500, VT510, VT520, VT525
VT LAN40

2.4.1 ASCII

ASCII obsahuje aj riadiace znaky. Zoznam implementovaných ASCII riadiacich inštrukcií použitých vo výslednom zariadení.

Názov	Hex	C-escape	Ctrl-Key	Popis
BEL	0x07	\a	∧G	Terminal Bell
BS	0x08	\b	∧H	Backspace
HT	0x09	\t	∧I	Horizontal TAB
LF	0x0A	\n	∧J	Linefeed (newline)
FF	0x0C	\f	∧L	Formfeed
CR	0x0D	\r	∧M	Carriage return

Tabulka 3: ASCII štandard

Sekvencia **BELL** sa v termináloch používa pre vyvolanie upozornenia prostredníctvom buzzeru (piezo meniča). V bakalárskej práci je sekvencia BELL implementovaná na uspatie/vypnutie LCD bez stráty údajov

Sekvencia **BS** nahradí posledný znak prvého textu prvým znakom druhého textu:
echo -e „Hello\bWorld“ → „HellWorld“

Sekvencia **HT** posunie kurzor na ďalšiu tabulátorovú pozíciu:
echo -e „Hello\tWorld“ → „Hello World“

Sekvencia **LF** posunie kurzor na nový riadok. Ak je riadok zaplnený, tak ho vyčistí.

Sekvencia **FF** stránkový posun:
echo -e „Hello\fWorld“ → „Hello
. World“

Sekvencia **CR** posunie kurzor na začiatok riadku:
echo -e „Hello\rIla“ → „Ilalo“

2.4.2 VT 100 štandard

Počet riadiacich inštrukcií štandardu VT100 je mnoho [12]. Pri implementácií som zvolil najpoužívanejšie a najužitočnejšie riadiace sekvencie pre prácu s textom na LCD. Zoznam implementovaných VT100 escape sekvencií, ktoré sú použité vo výslednom zariadení.

Esc-Key	Escape sekvencia	Názov
<ESC>[1J	'\e[1J'	Erase Up
<ESC>[2J	'\e[2J'	Erase Screen
<ESC>[7h	'\e[7h'	Enable line wrap
<ESC>[7l	'\e[7l'	Disable line wrap
<ESC>[{R};{C}f	'\e[1;5f	Force Cursor Position
<ESC>[5m	'\e[5m'	Turn blinking mode on
<Esc>E	'\eE'	Move to next line
<ESC>c	'\ec'	Reset default values
<ESC>[C	'\eC'	Cursor Forward
<ESC>[H	'\e[H	Cursor Home

Tabulka 4: Escape Sequences VT100

- **<ESC>[1J** - zruší text na všetkých riadkoch pred aktuálnym riadkom.
- **<ESC>[2J** - vyčistí text zo všetkých riadkoch.
- **<ESC>[7h** - povolí automatické zalomenie do nového riadku v prípade, že sa text do jedného nezmestí.
- **<ESC>[7l** - zruší automatické zalomenie do nového riadku. Text pretečie po 16. pozícií.
- **<ESC>[{R};{C}f** premiestni text na zvolený riadok R a zvolený stĺpec C. Ak číslo -riadok/stĺpec sú mimo rozsah, kurzor sa nastaví na posledný riadok a poslednú pozíciu.
- **<ESC>[5m** - povolí blikanie kurzoru.
- **<ESC>E** - presunie kurzor do nového riadku. Na poslednom riadku skroluje do prvého.
- **<ESC>[c** - resetuje všetky zmeny nastavené sekvenciami na defaultné hodnoty.
- **<ESC>[C** - posunie kurzor o jednu pozíciu vpred.
- **<ESC>[H** - nastaví kurzor na domovskú pozíciu - prvý riadok, nultá pozícia.

3 Popis modulov pre LCD terminálu

3.1 Požiadavky mikropočítača

Pri výbere MCU, čo je základná časť pre vývoj LCD terminálu, potrebujeme brať v úvahu požiadavky na procesor akými sú interný oscilátor, podpora USB rozhrania, podpora režimu OTG, dostatočný počet I/O na pripojenie LCD. Pri porovnaní sa zameriame na procesory Pic rady 24F. Ide o spoľahlivé MCU jednotky, ktorými disponuje univerzita.

- PIC® Microcontrollers:
  - **PIC24FJ32GB002**
  - **PIC24FJ32GB004**
  - **PIC24FJ64GB002**
  - **PIC24FJ64GB108**
  - **PIC24FJ64GC006**

Označenie	Interný Osc	OTG	USB	Prevedenie	Pins	Memory
24FJ32GB002	8 MHz, 32 kHz	Áno	Áno	QFN, SPDIP, SOIC, SSOP	28	8192 (B)
24FJ32GB004	8 MHz, 32 kHz	Áno	Áno	QFN, TQFP	44	8192 (B)
24FJ64GB002	8 MHz, 32 kHz	Áno	Áno	QFN, SPDIP, SOIC, SSOP	28	8192 (B)
24FJ64GB004	8 MHz, 32 kHz	Áno	Áno	QFN, TQFP	28	8192 (B)
24FJ64GB108	8 MHz, 32 kHz	Áno	Áno	QFN, TQFP	80	16384 (B)
24FJ64GC006	8 MHz, 32 kHz	Áno	Áno	QFN, TQFP	64	8192 (B)

Tabulka 5: Porovnanie jednotlivých druhov Pic

3.1.1 Výber procesora

Pri výbere MCU Unit potrebujeme taký druh procesora, ktorý nám umožní zapojiť LCD displej v 8 bitovom režime, nezávisle bude podporovať USB interface a umožňovať programovanie priamo na kontaktnom poli. Potrebujeme procesor s **11 I/O** na ovládanie displeja, **D+ / D-** pre vytvorenie USB interface a **PGED3 / PGEC3** pre vytvorenie ISP rozhrania. Keďže procesory rady **PIC24FJ*** majú na jednom pine mnoho funkčných režimov, ktoré sa nastavujú cez poistky, potrebujeme procesor s **15 I/O**. Pri práci na kontaktnom poli potrebujeme procesor v prevedení **DIP**, nie v prevedení **QFN, SOIC, TQFP**.

Z uvedených procesorov v prevedení DIP prichádzajú do úvahy PIC24FJ32GB002 (32 bitový) a PIC24FJ64GB002 (64 bitový) MCU, ktoré sú k dispozícií v prevedení SPDIP. Najideálnejším procesorom, z výberu prichádza do úvahy PIC24FJ64GB002

3.1.2 Popis procesora PIC24FJ64GB002

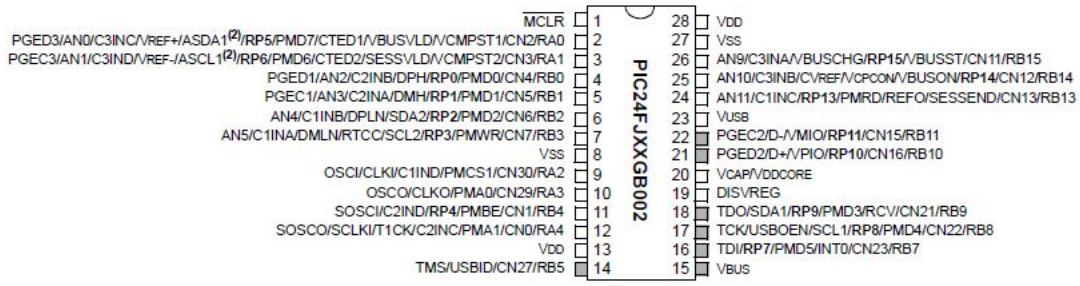
Ide o 16 bitovú, 32 MHz procesorovú jednotku [8]

Core Size	16-Bit
Rýchlosť	32 (MHZ)
Podporuje	I^2C, IrDA, SPI, UART/USART, USB OTG
Periférie	Brown-out Detect/Reset, LVD, POR, PWM, WDT
Počet I/O	21
Programová pamäť	64KB (22K x 24)
Typ Programovej pamäte	FLASH
Veľkosť RAM	8,192 (B)
Data Converters A/D	9x10b
Vnútorný Oscilátor	Áno
Typ vnútorného oscilátoru	8 MHz, 32 kHz
Možnosť napojiť externého Oscilátoru	Áno
Počet pinov / Konštrukčné prevedenie	28 / SPDIP
Prevádzková teplota	-40° C - 85° C
Prevádzkové napätie (Vcc/Vdd)	2 V - 3.6 V

Tabulka 6: Rozpis parametrov

Ďalšie prednosti MCU:

- Podpora rýchlosti low - speed a full - speed v režimu Host (Master).
- Podpora rýchlosti full - speed v režimu (Slave).
- 16 obojsmerných endpointov
- Registre portov: PORTA → (výstupy z portu) RA0 - RA4
- Registre portov: PORTB → (výstupy z portu) RB0 - RB5, RB7 - RB11, RB13 - RB15
- Ďalšie registre: TRISA, LATA, TRISB, LATB



Obrázek 5: Diagram pinov

3.1.3 Konfigurácia MCU PIC24FJ64GB002

MCU nakonfigurujeme prostredníctvom **poistiek** procesora. Pomocou poistiek sa nastavia vnútorné jednotky a módy, v ktorých má MCU pracovať. Taktiež konfiguračnými bitmi nastavujeme obmedzenie, či prípadné vypnutie modulov, ktoré v procesoru nevyužívame, tak ako aj zabezpečenie čítania, prístupu k pamäti vrátanie možnosti dolaďovania SW vybavenia MCU. Konfiguračné bity umožňuje programátor PRESTO [9] definovať priamo v užívateľskom rozhraní programu, avšak umožňuje bitovú konfiguráciu načítavať priamo z hex súboru. Pri spustení kompilátora sa zdrojový kód preloží do hex súboru, s ktorým programátor pracuje.

Pri vývoji, podľa diagramu pinov 5, si určíme, aké piny na čo použijeme.
Porty **MCLR**, RA0/**PGED3**, RA1/**PGEC3** použijeme pre **ISP** rozhranie.
Port **RB0** použijeme pre kontrolnú LED diódu.
Porty **RA0**, **RA2** použijeme ako ovládacie porty LCD. Porty **RB1**, **RB2**, **RB3**, **RB7**, **RB8**, **RB9**, **RB13**, **RB14** použijeme ako 8 bitovú, dátovú zbernicu pre LCD.
Porty RB10/**D+**, RB11/**D-** využijeme ako dátové vodiče USB, port **VBus** zapojíme ako detekciu USB napájanie

Port RA0 využívame primárne ako súčasť ISP rozhranie a sekundárne ako riadiaci pin pre LCD. Vzhľadom k tomu, že programátor umožňuje ISP programming, MCU je pri programovaní v stave konfigurácie a LCD ho využíva ako riadiaci, tj. ako OUTPUT, programovanie nebude vykazovať žiadne problémy. Radič LCD navyše príma len potvrdzovacie signály v trvaní cca. **1ms** → **log. "1"**→ **2ms** →**log. "0"**→ **1ms**. Ostatné ignoruje.

Problém by nastal, keby pin port RA0 slúžil ako INPUT, a v čase programovania by sa na vstup z vonkajšieho zdroja dostal impulz log. "1".

Pri zostavovaní zariadenia ako terminál, požadujeme, aby vedel komunikovať cez USB ako Master v režime CDC [5]. Využijeme priamo 4MHz výstup z interného oscilátora. Konfiguráciou konfiguračného bitu **FNOSC_FRCPLL** vyberieme vnútorný oscilátor **FRC** (Fast-Rc-Oscilator) + **PPL** (Phase-Locked-Loop). Taktovacia frekvencia MCU sa nastaví

zmenou multiplikátora a deliacej hodnoty, ktoré riadia MCU clock.

Konfiguračným bitom **PLL96MHZ_ON** 96 MHz PLL povolíme automatický štart v MCU. Deliacu hodnotu nastavíme ako "No Div"bitom **PLLDIV_NODIV**. Oscilátor vstup využíva priamo ako 4MHz input. Nastavením bitov **POSCMOD_NONE** zakážeme hlavný oscilátor, vypneme hodinový prepínač **FCKSM_CSDCMD**, **OSCIOFNC_ON** nastavíme port **RA3** ako I/O - disablujeme defaultný mód portu ako OSCO (Clock Output).

SOSCSEL_IO zrušíme defaultný mód portov RB4, RA4 sekundárneho oscilátora, ktoré sú v stave (SOSCO, SOSCI) a nastavíme ich ako I/O.

Kompletná realizácia je v kóde 3. Všetky ostatné nepopísané konfiguračné bity vychádzajú z doporučeného zapojenia MCU. Podrobný popis hlavných konfiguračných bitov:

- _CONFIG1:
  - **FWDTEN_OFF** - Watch dog timer - disabled
  - **WINDIS_OFF** - Windowed WDT - disabled
  - **WDTPS_PS1** - WDT postscale - 1:1
  - **FWPSA_PR32** WDT prescale - 1:32
  - **ICS_PGx3** - Nastavenie rozhrania ISP pre PGEC3 PGED3
  - **JTAGEN_OFF** - Dovoľuje prístup k ladiacemu modulu na MCU - disabled
  - **GWRP_OFF** - (Write protect) Ochrana proti zápisu - disabled
  - **GCP_OFF** - (Code protect) Ochrana proti čítaniu kódu - disabled
- _CONFIG2:
  - **FNOSC_FRCPL** - Výber vnútorného oscilátoru FRC
  - **OSCIOFNC_ON** - Disablovanie portu RA3 ako OSCO (clock output)
  - **FCKSM_CSDCMD** - Clock switch and monitor - disabled
  - **PLL96MHZ_ON** - 96 MHz PLL - enabled
  - **POSCMOD_NONE** - Hlavný Oscilátor - disabled
  - **PLLDIV_NODIV** - Oscilátor input - 4MHz
- _CONFIG3:
  - **SOSCSEL_IO** – Sekundárny Oscilátor - disabled
  - **WUTSEL_LEG** – Legacy Wake-up Timer
  - **WPDIS_WPDIS** – Segmented code protection - disabled

Firmware MCU ďalej nastavuje ďalšie konfiguračné poistky, zdrojový kód 3.

```
#if defined (__USB_CDC_DEV_D__)
    _CONFIG1(WDTPS_PS1
            & FWPSA_PR32
            & WINDIS_OFF
            & FWDTEN_OFF
            & ICS_PGx3
            & GWRP_OFF
            & GCP_OFF
            & JTAGEN_OFF)
    _CONFIG2(POSCMOD_NONE
            & I2C1SEL_PRI
            & IOL1WAY_OFF
            & OSCIOFNC_ON
            & FCKSM_CSDCMD
            & FNOSC_FRCPLL
            & PLL96MHZ_ON
            & PLLDIV_NODIV
            & IESO_ON)
    _CONFIG3(WPFP_WPFP0
            & SOSCSEL_IO
            & WUTSEL_LEG
            & WPDIS_WPDIS
            & WPCFG_WPCFGDIS
            & WPEND_WPENDMEM)
    _CONFIG4(DSWDTPS_DSWDTPS3
            & DSWDTOSC_LPRC
            & RTCOSC_SOSC
            & DSBOREN_OFF
            & DSWDTEN_OFF)
#elif defined(__LCD_DIALL_SET__)
    _CONFIG1(JTAGEN_OFF
            & ICS_PGx1
            & FWDTEN_OFF)
    _CONFIG2(FNOSC_FRC)
```

Výpis 3: Kompletné nastavenie poistiek

Vývoj terminálu budeme realizovať prostredníctvom programátoru Presto [9]. Ide o rýchly USB programátor podporujúci ISP (In-System-Programming) programovanie MCU Unit zapojeného priamo na doske. Prototypové zapojenie budeme vyvíjať na kontaktnom poli. Ako vývojove prostredie použijeme MPLAB IDE 8 [6] spolu s kompilerom C30 C [7]

3.2 Výber LCD displeja

Ako LCD displej som si zvolil typ **PVC160203PGL01**, ktorý som mal k dispozícii z vlastných zdrojov.

LCD displej je klasický LCD [10] 2x16 znakov bez podsvietenia.
Napájanie : od 4.7 - 5.3 V.
Prúd: 2.5 mA.

LCD obsahuje HD44780 radič poskytujúci znakovú sadu pre LCD. Ide o LCD, ktorý je výkonnosťou najprijateľnejší a za rozumnú cenu.

LCD umožňuje dva módy prijímania dát. **4 bitový** a **8 bitový**. Dáta sa odosielajú prostredníctvom 4 resp. 8 bitovej zbernice. LCM (Load Control Module) modul ďalej podporuje dva režimy **znakový** a **riadiaci** .

Pri Riadiacom režime LCM modul spracuje vstupné signály na 8 bitovej zbernici a vykoná ich definovanú funkcionalitu, tabuľka 7 napr. posunutie kurzora, odstránenie znakov na displeji a podobne. Pri oživovaní musí byť LCD v riadiacom režime a oživí sa po odosielaní sekvencií riadiacich impulzov, kapitola 4.1. Pri dátovom režime, LCM spracuje vstupné dáta na 8 bitovej zbernici a zobrazí definovaný znak .

Jednotlivé režimy a módy sa nastavujú prostredníctvom troch riadiacích vstupov LCD displeja **E**, **RW**, **RS**.

Každý jeden z týchto vstupov, môže byť v stave log. "0"alebo log. "1".

Vstupom **RW** (Read/Write) nastavujeme LCM do stavu **log. "0"→ Write**, zapisovania znakov do DDRAM, **log. "1"→ Read** - čítanie znakov z DDRAM.

Vstupom **RS** nastavujeme LCM modul do stavu **log. "0"→ Command** - LCM po prijatí dát na dátovej zbernici ich vyhodnotí ako inštrukcie a vykoná definovanú funkcionalitu, **log. "0"→ Dáta** - LCM prijaté dáta berie ako znaky a zobrazí ich.

Vstup **E** slúži ako potvrdzovací. Odoslané dáta do LCD je nutné v určitú chvíľu potvrdiť. Pre tento druh LCD, je oneskorenie nastavenia binárnych stavov: **log. "1"→ 1us → log "0"**

Odosielanie dát v znakovom a riadiacom režime bude chronologicky:

Z [(set S)= log "0"] - [send Data] - [(set E) = log. "1"→ log "0"]
R [(set S)= log "1"] - [send Data] - [(set E) = log. "1"→ log "0"]

3.2.1 Príkazová sada LCD

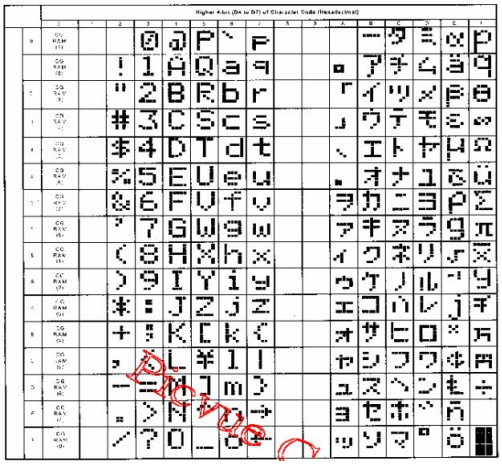
Príkazovou sadou LCD [10] sme schopný ovládať LCD. Sada nám umožňuje, operácie s kurzorom ako nastavenie na **domovskú pozíciu**, **nastavenie kurzoru na daný riadok a danú pozíciu**, **pohyb kurzoru vpred**, **povolenie/zakázanie blikania kurzoru**, **uspanie displeja**,**zmazanie riadku displeja**

Inštrukcia	Hex
Povolenie Displeja	0x0F
Zakázanie Displeja	0x08
Zmazanie všetkých znakov na Displej	0x01
Zobrazenie kurzora	0x02
Vypnutie kurzora	0x0C
Oživenie len prvého riadku (vypnutie druhého) LCD	0x30
Oživenie (zapnutie) oboch riadkov LCD	0x38
Prepnutie prijímania dát z 8 bitovej na 4 bitovú zbernicu)	0x20
Nastavenie kurzora na prvý riadok	0x80
Nastavenie kurzora na druhý riadok	0xC0
Nastavenie kurzora na prvý riadok na 8 pozíciu	0x80 + 0x08

Tabulka 7: Príkazová sada LCD

3.2.2 Znaková sada LCD

Z obrázka znakovej sady LCD [10] vidíme, že pri prímaní znakov z počítača potrebujeme znaky od 0x20 - 0x7F.



Obrázek 6: Znaková sada HD44780I

3.2.3 Oživenie displeja

Pri zapnutí napájania displeja, sa LCM modul nachádza v polo-disablovanom (inicializačnom móde). V tejto fáze modul sám o sebe nedokáže spracovávať dáta na zobrazenie.

Z LCD pinov na riadiacej zbernici (E, R/W, RS) modul akceptuje len logické hodnoty E → log.1/0, R/W → log.0, RS → log.1.

Neakceptuje R/W → log.1 (čítanie displeja) a RS → log.0 (znakový režim). Očakáva sekvenciu riadiacich znakov z tabuľky 7 odoslané v určitom poradí a s určitým oneskorením. Po obdŕžaní riadiacich inštrukcií, sa LCM prepne do plne funkčného režimu v ktorom povolí ostatnú funkcionalitu LCD.

```
Delay_ms(100);
LCD_Send_(1,LCD.bit8_1ine1,null,0, null);
Delay_ms(5);
LCD_Send_(1,LCD.bit8_1ine1,null,0, null);
Delay_us(150);
LCD_Send_(1,LCD.bit8_1ine1,null,0, null);
Delay_us(150);
LCD_Send_(1,LCD.bit8_line2,null,0, null);
Delay_us(150);
LCD_Send_(1,LCD.disable_lcd,null,0, null);
Delay_ms(3);
LCD_Send_(1,LCD.clear_lcd,null,0, null);
Delay_ms(15);
```

Výpis 4: Oživenie displeja - výpis z kódu

3.2.4 Vypisovanie textu na displej

Po nastavení riadiaceho vstupu RS na log. 0, sa procesorom odošlú dáta na dátovú zbernicu, porty procesora: RB1, RB2, RB3, RB7, RB8, RB9, RB13, RB14. V okamihu odoslania dát zbernicou sa po 1ms nastaví riadiaci vstup LCD E na log. 1. Po 2ms sa vstup E nastaví na log. "0"a na 1ms sa procesor uspí. Vo fáze odosielania dát po zbernici LCM displeja nereaguje. Dáta na dátovej zbernici spracuje až vo chvíli, kedy sa s oneskorením na vstup E privedie log. 1 a následne log. 0. V tomto kroku LCM dostáva impulz, aby prijal dáta na dátovej zbernici. Akonáhle ich príjme, tak sa znovu prepne do režimu "waiting", kedy čaká na potvrdzovací impulz na vstupe E. Všeobecne platí, že pokiaľ LCD hneď po odoslaní dát nedostane potvrdzovací impulz, tak hladiny logických impulzov na dátovej zbernici ignoruje. Hneď po potvrdzovacom impulze E je nutné, aby procesor počkal 1ms. LCM je pomalší než MCU, a pri odosielaní by LCM modul dostal potvrdzovací impulz ešte skôr, než by stihol vypísať predošlé dáta a znaky by sa nevypísali.

Znaky vypisujeme prostredníctvom funkcie, ktorá musí obsahovať okrem iných parametrov parameter vyjadrujúci počet odosielaných dát na LCD displej, a parameter reprezentujúci samostatné dáta. Počet prenášaných dá využívame pri odosielaní dát na LCD prijatých z USB. Samotné dáta sa prenášajú ako **byte/uint8_t** prípadne ako **string** (text). Parameter funkcie je ako **unsigned char** $*$ **dáta** čo je pointer dátového typu unsigned char odkazujúci do pamäte procesora - na miesto v pamäti, na ktorého adrese sa nachádzajú dáta.

Byte reprezentuje 8 bitov, čo je 0-255. Pointer typu unsigned char odkazuje na adresu v pamäti, ktorá zaberá 8 bitov = 1 byte.

3.2.5 Zápis dát

Porty procesora sú reprezentované ako register, ktorý ovláda jednotlivé piny. Pri odosielaní dát po zbernici, môže byť v jednom bitu registra log. 1 alebo log. 0. Do vybraného bitu registra môžeme nastaviť logickú úroveň. Problém nastáva, ak chceme ovládať bity registru na rôznej pozícií. Pri odosielaní 8 bitov 11111111, by sa ovládali jednotlivé bity vždy len vedľa seba. Ak chceme ovládať bity na pozícií 14,13 , musíme 11111111 prehnať maskou 11000000 a výsledok posunúť na 7 pozíciu bitového registra. Pri ovládaní pozícii 11,10,9 bude maska 00111000 a výsledok posunutý bitovým posunom na 4 pozíciu bitového registra.

Ukážka buniek s pozíciami, priradenými portmi (RB), ktoré ovládajú a ich zapojenie na LCD zbernicu (D0-D7)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B15	B14	B13	-	B11	B10	B9	B8	B7	-	B5	B4	B3	B2	R1	R0
	D7	D6	-			D5	D4	D3	-			D2	D1	D0	

Tabulka 8: Rozmiestnenie bitov registra PORTB

Porty RB6 a RB12 procesor nemá zapojené. Pri používaní zvolených portov pre dátovú zbernicu, použijeme prvú masku 0xC0 s bitovým posunom 7, druhú masku 0x38 s bitovým posunom 4 a tretiu masku 0x07 s bitovým posunom 1.

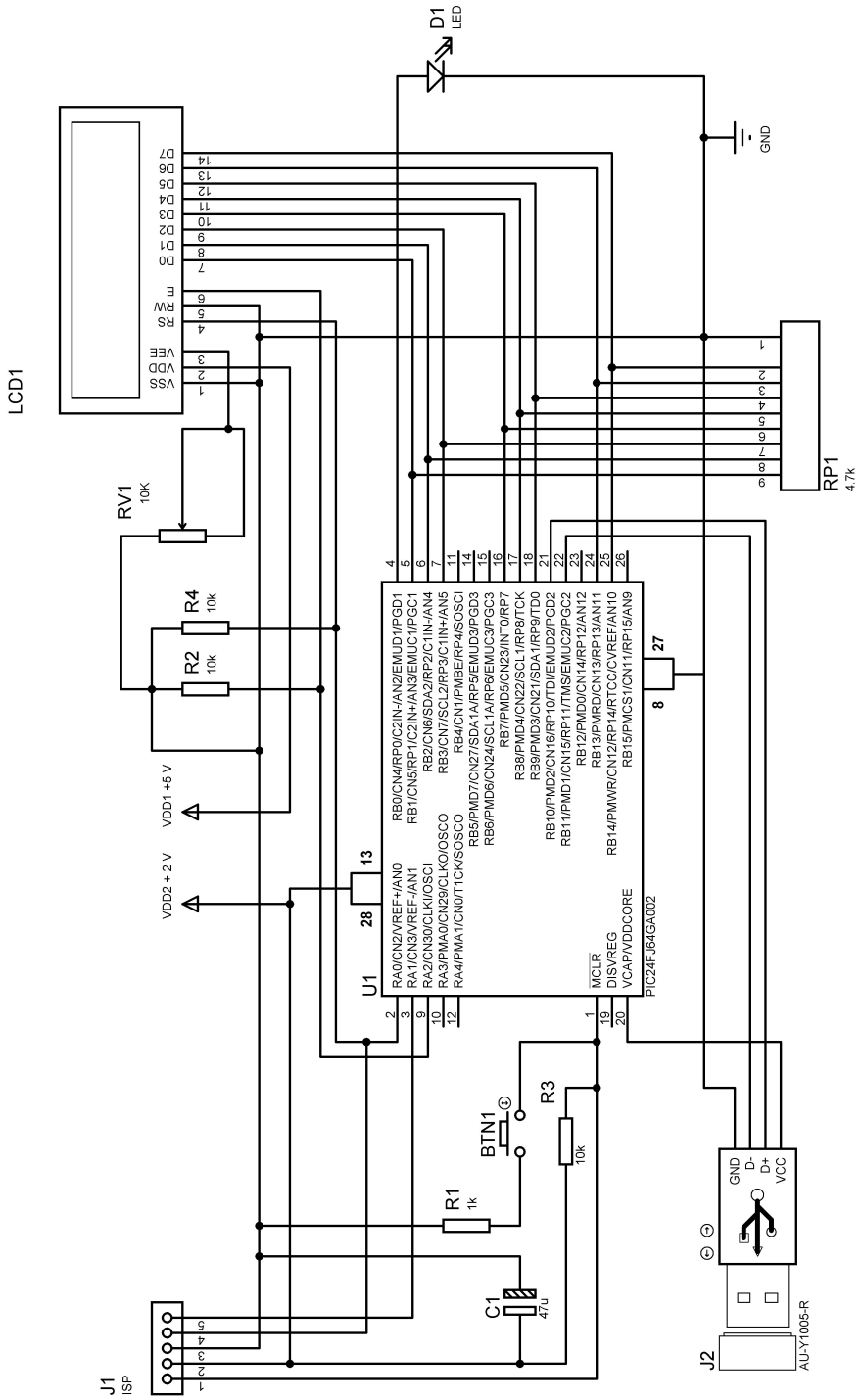
4 Návrh zariadenia

Schéma zapojenia LCD terminálu je na obrázku 7.

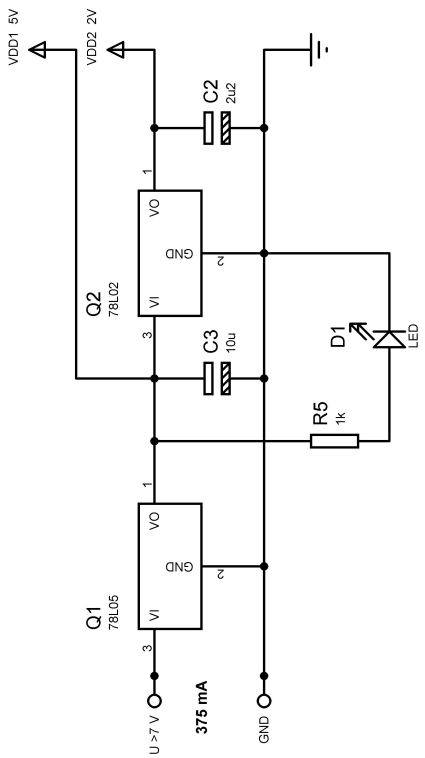
Riadiaca jednotka U1 je pinmi 2, 3 ako PGED3/PGEC3 pripojená na ISP rozhranie. PGED sa využíva ako dátový vstup/výstup, PGEC ako vstupný hodinový impulz Clock programátora. Reset MCU MLRC je na rozhranie ISP privedené ako VPP. Napájanie ISP je pripojené na lokálne napájanie VDD2. Tlačidlo BTN1 je pripojené k MCLR procesora ako reset privedené ku GND rezistorom R1. Na odpore R1 vznikne úbytok napätia a zabráni sa poškodeniu programátora. Rezistor R3 je zapojený ako pull-down rezistor na udržanie log. 0 na porte MCLR. Ako filtračný člen medzi VDD2 a GND je použitý kondenzátor C1. Dátové vodiče USB rozhrania D+ a D- sú privedené na I/O procesora RB10/RB11. Procesor U1 má na týchto portoch implementované vnútorné pull-down rezistory, preto ďalšie použitie pull-down rezistorov nie je potrebné.

Displej LCD1 je pripojený pomocou dátovej zbernice na dátových vstupov D0 - D7 k I/O portom procesora RB1, RB2, RB3, RB7, RB8, RB9, RB13, RB14. Na dátovej zbernici sú použité pull-down rezistory na udržanie napäťovej úrovne log. 0 pri stave neaktivity, v podobe rezistorového poľa RP1. Riadiace vstupy LCD1 RS, E sú pripojené na I/O porty procesora RA0, RA2. Riadiaci pin R/W, môžeme pripojiť na GND, nakoľko pri zápise dát na LCD musí byť na ňom log. 0 a čítanie z LCD nevyžadujeme. Riadiace piny E, RS sú pomocou rezistorov R2, R4 privedené na GND na udržanie log. 0 pri neaktivite. Trimrom RV1, pripojeného medzi GND a VEE LCD1, regulujeme úroveň jasu displeja. Signalizačná dioda D1 je pripojená na port procesora RB0. Piny 8, 27 procesora U1 sú pripojené na GND, piny 28, 13 k napájaciemu napätiu VDD2. LCD1 je pripojený k napätiu VDD1 5 V.

Zdroj LCD terminálu, schéma [8], je tvorený dvomi stabilizačnými modulmi Q1 a Q2, ktoré sú zapojené do série. Vstupné napájacie napätie stabilizátor Q1 ustabilizuje na VDD1 5V, ktoré je použité na napájanie LCD1. Stabilizátor Q2 ustabilizuje napätie VDD1 na napätie VDD2 2V, ktoré napája procesorovú jednotku. Kondenzátory C2, C3 sú zapojené ako filtračné prvky napätí VDD1 a VDD2. Signalizačná dioda D2 je v sérii s rezistorom R5 zapojená paralelne medzi VDD1 a GND. Ako zdroj napájania je použitý ľubovoľný, spínavý zdroj 7 V, 375mA.



Obrázek 7: Návrh zapojenia LCD terminálu



Obrázek 8: Návrh zapojenia zdroja

4.1 Oživenie zariadenia

Po zrealizovaní zapojenia napojíme LCD terminál na zdroj el. napätia. V prípade, že displej nereaguje, nastavíme mu trimrom RV1 požadovanú hodnotu jasu. Na ISP konektor pripojíme programátor Presto, ktorým MCU naprogramujeme.

Po úspešnom naprogramovaní sa MCU resetuje. Pri štarte programu v MCU sa všetky moduly a periféria inicializujú. Pri správnom inite signalizačná dioda D1 raz blikne. Pokiaľ došlo k chybe inicializácie LCD, D1 dva krát na 1s blikne. Ak došlo k nesprávnemu inicializovaniu USB rozhrania, D1 blikne tri krát. Pri inite sa na LCD vypisujú názvy modulov, ktoré sa aktuálne inicializujú spolu s informáciou o stave inicializovania. Po načítaní všetkých modulov, MCU vypíše na LCD blikajúcu správu „READY“ po ktorej je možné pripojiť zariadenie do USB. Pokiaľ je zariadenie z USB odpojené, na displej sa vypíše „PC - Odpojeny !“. Po zapojení zariadenia do USB sa na displej vypíše
správa „PC - Pripojeny !“ a po necelej sekunde sa obsah displeja vyčistí. Pokiaľ je terminál zapojený a pripravený prímať dáta, signalizačná D1 bliká.

Pri návrhu schémového zapojenia USB terminálu som využil grafický software Proteus [11]

4.2 Zoznam súčiastok

Označenie	Hodnota
C1	47 μF
C2	2.2 μF
C3	10 μF
R1, R5	1kΩ
R2, R3, R4	10 kΩ
RP1	4.7 k Ω
RV1	10 kΩ
U1	PIC24FJ64GB002
Q1	78L05
Q2	78L02
LCD1	PVC160203PGL01
D1	LED 3mm čerená
D2	LED 3mm zelená
J1	Kon. Kolíková lišta 1 x 5 pin / 2,54mm
J2	Kon. Kolíková lišta 1 x 4 pin / 2,54mm
Btn1	5mm TS045-50
Napájací zdroj	SPN5038A IN 100/240, Out > 7V/375 mA

Tabulka 9: Zoznam použitých súčiastok

5 Programové rozhranie mikropočítača

V tejto časti je popísaný návrh jednotlivých modulov programového rozhrania mikropočítača.

5.1 Modul prijímania dát z počítača

O samotné príjmanie dát prostredníctvom USB sa stará firmware MCU. Samotné firmware obsahuje dve funkcie **getsUSBUSART** a **putsUSBUSART**.

Funkcia getsUSBUSART slúži na získanie prenesených dát.
Funkcia getsUSBUSART(byte, int) má dva vstupné parametre typu byte a integer. Prvým parametrom definujeme pole buffer, do ktorého sa majú dáta ukladať a druhým definujeme počet dát, ktoré sa majú prenášať. Návratový typ funkcie je integer a reprezentuje počet dát, ktoré boli z počítača prenesené. V pôvodnej realizácií firmware, getsUSBUSART a putsUSBUSART bežia pod metodou **ProcessIO()** v cykle. Akonáhle sú dáta prenášané (to znamená, že návratová hodnota funkcie getsUSBUSART je nenulová), prenesené dáta sa odošlú funkciou putsUSBUSART znovu do počítača.

V našom návrhu metodu putsUSBUSART vynecháme a navrhneme metodu **LCD_Send_()**, ktorá nám dáta spracuje.
Pokiaľ prenos nie je nadviazaný, prípadne, zariadenie nie je na zbernici ešte inicializované, tak je naplnená podmienka

```
if((USBDeviceState < CONFIGURED\_STATE)||(USBSuspendControl==1)) return;
```

a obsah metody PocessIO sa nevykoná. V opačnom prípade sa zisťuje, či sú do kontroléru odosielané dáta, premenná **USB_Received.size** obsahuje počet prenesených dát. Ak je hodnota nenulová, do premennej **USB_Received.ready** sa pridá príznak 1, ktorý hovorí cyklu, aby čakal kým sa všetky dáta prenesú do kontroléra. Dáta sa z funkcie getsUSBUSART najprv uložia do dočasného bufferu **USB_Transmision_Receive.Tmp.Data** odkial sa uložia do dátového bufferu. Následne sa obsah dátového bufferu odošle funkciou LCD_Send_(), kde sa spracuje. Pri prenose funkciou LCD_Send_() prenášame aj počet prenesených znakov.

```
void ProcessIO(void)
{
  BlinkUSBStatus();
  if((USBDeviceState < CONFIGURED_STATE)||(USBSuspendControl==1)) return;
  if(USBUSARTIsTxTrfReady())
    {
      if (USB_Received.ready == 0)
        {
          USB_Received.size = getsUSBUSART(USB_Transmision_Receive.Tmp.Data,
              USB_Transmision_Receive.Tmp.buffer_size);

        if (USB_Received.size > 0)
```

```
            {
                BYTE cdc_rx_len;
                for(cdc_rx_len = 0; cdc_rx_len < USB_Received.size; cdc_rx_len++)
                    {
                        if(Keys.Tmp.swt == 2)
                          {
                            USB_Received.Data.Command[cdc_rx_len] =
                                USB_Transmision_Receive.Tmp.Data[cdc_rx_len];
                          } else {
                                        USB_Received.Data.Data[cdc_rx_len] =
                                            USB_Transmision_Receive.Tmp.Data[cdc_rx_len];
                                    }
                    }
                 USB_Received.ready = 1;
                USB_Transmision_Receive.cp_ptr = 0;
             }
        }
         if (USB_Received.ready)
           {
              BlinkUSBStatus();
              LCD_Send_(3, null,USB_Received.Data.Data, null, USB_Received.size);
           }

         if (USB_Received.ready)
           {
              USB_Received.ready = 0;
              USB_Received.size = 0;
              USB_Transmision_Receive.cp_ptr = 0;

              for(cdc_rx_len = 0; cdc_rx_len < CDC_DATA_IN_EP_SIZE; cdc_rx_len++)
                    {
                       USB_Received.Data.Data[cdc_rx_len] = 0;
                       USB_Received.Data.Command[cdc_rx_len] = 0;
                       USB_Transmision_Receive.Tmp.Data[cdc_rx_len] = 0;
                    }
             }
     }
    CDCTxService();
}
```

Výpis 5: Návrh funkcie pre príjem dát

5.2 Makra ovládania registrov

Návrh makier ktoré ovládajú jednotlivé registre a vykonávajú operácie nad nimi. Funkcie končiace na „_“ napr. Bit_Set_ berú ako parameter _set_bit_xxx_ označenie registra PORT / TRIS, napr. PORT**A**, PORT**B**. Ostatné berú ako parameter _set_bit_xxx_ názov registra, napr. **PORTA**, **PORTB**, **TRISB**. Parametrom **___pin___** sa udáva číslo pozície daného pinu v danom registre. Parametrom **___position___** sa udáva bitový posun.

```
#define   Bit_Set_(_set_bit_on_,___pin___)  PORT ## _set_bit_on_ |= 1<<___pin___
#define   Bit_Clr_(_set_bit_clr_,___pin___)  PORT ## _set_bit_clr_ &= ~(1<<___pin___)
#define   Bit_Toogle_(_set_bit_tgl_,___pin___) PORT ## _set_bit_tgl_ ^= (1<<___pin___);
#define   Bit_Set(_set_bit_on_,___pin___)     _set_bit_on_ |= 1<<___pin___
#define   Bit_Clr(_set_bit_clr_,___pin___)    _set_bit_clr_ &= ~(1<<___pin___)
#define   Bit_Toogle(_set_bit_tgl_,___pin___) _set_bit_tgl_ ^= (1<<___pin___);
#define   Config_IN_(_set_bit_in_,___pin___) TRIS ## _set_bit_in_ |= 1<<___pin___
#define   Config_OUT_(_set_bit_out_,___pin___) TRIS ## _set_bit_out_ &= ~(1<<___pin___)
#define   Config_IN(_set_bit_in_,___pin___)   _set_bit_in_ |= 1<<___pin___
#define   Config_OUT(_set_bit_out_,___pin___) _set_bit_out_ &= ~(1<<___pin___)
#define   AD_Set_Digital(___pin___)           AD1PCFG |= 1<<___pin___
#define   AD_Set_Analog(___pin___)            AD1PCFG &= ~(1<<___pin___)
#define   Set_Pin(_set_bit_out_,___pin___,___position___) _set_bit_out_ = ___pin___<<
    ___position___
#define   Set_On(___set_me___) ___set_me___ = ON
#define   Set_Off(___set_me___) ___set_me___ = OFF
#define   Write_Pin(_set_bit_,___pin___) _set_bit_ = ___pin___ << 7
```

Výpis 6: Makra pre prácu nad registrami

5.3 Modul zápisu dát na LCD

Funkcia zápisu **Write** ako taká, je viditeľná iba v rámci zdrojového kódu LCD. Je to funkcia, ktorá je volaná z funkcie LCD_Send_(...). Funkcia Write riadi registre MCU a dáta sa priamo odosielajú na dátovú zbernicu MCU. Pri odosielaní dát je nutné použiť bitové posuny, kapitola 3.2.5.
Funkcia obsahuje dva parametre **mode**, **___data___**. Parametrom mode sa určuje či sú dáta znaky alebo riadiace frekvencie, kapitola 3.2.3, tabuľka 7.

```
#define Write(x,y)  _sd_s12_(x,y)

PRIVATE void _sd_s12_(int mode, uint8_t ___data___) {
      int  i  = 0;
      if (i  == 1) {___data___ = LCD.port_clr;}
              *LCD.Data.Port = (___data___ & LCD.maskA) << LCD.shift_A;
              *LCD.Data.Port |= (___data___ & LCD.maskB) << LCD.shift_B;
              *LCD.Data.Port |= (___data___ & LCD.maskC) << LCD.shift_C;

            if (mode == 0 && i < 1) { Bit_Set(*LCD.Direct.Port,LCD.Direct.RS); }
            if (i  < 1) { LCD_GO; }
            if (mode == 0 && i < 1) { Bit_Clr(*LCD.Direct.Port,LCD.Direct.RS); }

              *LCD.Data.Port &= (LCD.port_clr & LCD.maskA) << LCD.shift_A;
              *LCD.Data.Port &= (LCD.port_clr & LCD.maskB) << LCD.shift_B;
              *LCD.Data.Port &= (LCD.port_clr & LCD.maskC) << LCD.shift_C;
  }
#undef Write
```

Výpis 7: Návrh funkcie zápisu dát do LCD

5.4 Modul spracovania dát

Funkcia LCD_Send_(...) slúži na spracovanie prijatých dát z počitača. všetky prijaté dáta sú odoslané do tejto funkcie ktorá ich porovná so svojifmi štandardmi terminálu. Následne ich odošle na LCD displej, prípadne pokiaľ ide o riadiace sekvencie, vykoná ich funkcionalitu. Identifikácia riadiacich príkazov je implementovaná ako automat, ktorý pri každej prijatej sekvencií znakov, hľadá v nich definované sekvencie príkazov command. Pokiaľ ich nájde, spracuje ich.

```
while((*___varchar_data___ && ___size_character___ == null) || (*___varchar_data___ &&
      ___size_character___ != null && char_len <= ___size_character___ ))
{
int  is_valid  = 1;
is_valid  &= *___varchar_data___ >= 0x20;
is_valid  &= *___varchar_data___ <= 0x7F;
is_valid  &= *___varchar_data___ != 0x1B;
is_valid  &= vt_enable == 0;
is_valid  &= vt_mode == 0;
is_valid  &= vt_mode_ == 0;

if (  is_valid  )
  {
      character++;
      if (tmp_line !=0) { tmp_line++;}
      if  (character  >= 16 || (tmp_line !=0 && tmp_line >=16))
         {character  = 0; tmp_line=0;
          if (wrap_disabled ==0)
            { LCD_GoTo(2,0, null, null, null );}
         }
      Write(0, *___varchar_data___);
      ___varchar_data___++;

  } else {
     #ifdef ___VT_100_SET___
   int  i ;
           if (*___varchar_data___ == Standards.VT100.carriage) // '\r' Carriage
             {...}

           if (*___varchar_data___ == Standards.VT100.form_feed) // '\f' Formfeed
             {...}

           if (*___varchar_data___ == Standards.VT100.bell) // '\a' Terminal bell
             {...}
           if (*___varchar_data___ == Standards.VT100.linefeed) // '\n' Linefeed
             {...}

           if (*___varchar_data___ == Standards.VT100.horizontal_tab) // "\t" Horizontal tabulator
             {...}

           if (*___varchar_data___ == Standards.VT100.backspace) // '\b' Backspace
             {...}
```

```
if (*___varchar_data___ == 0x1B)
  {
    vt_enable = 1;
    ___varchar_data___++;
    if (*___varchar_data___ == 0x5b)
      {
        vt_mode = 1;
        disp_poz = 1;
        ___varchar_data___++;
        line2  = *___varchar_data___;
        ___varchar_data___++;

          if (*___varchar_data___ == 0x3b) // '\e[x;yf' Cursor Moving position
            {...
            if (*___varchar_data___ == 0x66) {...} else {...} // '\e[2;2f'
            if (*___varchar_data___ == 0x66) {...} else {...} // '\e[2;16f'

                } ese {
          if  (*___varchar_data___ == 0x43) // '\e[C' Forward Cursor
                {...}    else

          if  (*___varchar_data___ == 0x48) // '\e[H' Home position
                {...}}
          }
  } else
    if  (*___varchar_data___ == 0x63) // '\ec' Reset to Init values
      {...}
    if  (*___varchar_data___ == 0x45) // '\eE' Move to next line
      {...}

if (disp_poz == 1 )
  {
    if (*___varchar_data___ == 0x35) // '\e[5m' Blink
        if (*___varchar_data___ == 0x6D)
          {...}

    if (*___varchar_data___ == 0x32) // '\e[2J' Erase displej
      {...}
    if (*___varchar_data___ == 0x31) // '\e[1J' Clear lines before  text
      {...}

    if (*___varchar_data___ == 0x37) // '\e[7*' Line Wrap
      {...
        if (*___varchar_data___ == 0x68) // '\e[7h' Enable Line Wrap
          {...}
        if (*___varchar_data___ == 0x6C) // '\e[7l' Disable Line Wrap
          {...}
      }
}
```

Výpis 8: Návrh funkcie spracovania dát

6 Stabilita zariadenia a spoľahlivosť

Zariadenie textového terminálu bolo navrhnuté s čo najmenším použitím šúčiastok. Použitím procesora nie je nutné zariadenie obstarávať neakými modulmi navyše, čo ma za následok minimálnu poruchovateľnosť zariadenia.

Pri navrhnutom SW riešení MCU, terminál nevykazoval žiadne známky chýb ako samovoľných resetov, mrznutí, prípadne kolabsov zapríčinených pretečením vnútorných bufferov. Celkový čas od spustenia terminálu až po použiteľný mód prijímania dát je 10s.

Pri prenose neboli pozorované prípadné stráty prenášaných sekvencií alebo znakov.

6.1 Nasadenie do systému UNIX

Terminál bol nasadený a testovaný pod operačným systémom DEBIAN.

Zariadenie sa po pripojení do USB správa ako zariadenie pripojené prostredníctvom RS-232 a pod systémom DEBIAN je viditeľné ako **/dev/ttyACM0** .

Vylistovanie pripojeného zariadenia urobíme príkazom : **dmesg | grep tty** .
Vo výpise sa objaví „cdc_acm 2-2.2 :1.0: ttyACM0: USB ACM device“ .

Po pripojení není nutné terminál nijak konfigurovať v „ /etc/inittab“
Po pripojení stačí odoslať dáta prostredníctvom terminálu v DEBIANE.

Napríklad:

```
echo -e 'Ahoj ' > /dev/ttyACM0

echo -e 'AhojSvet\rCau' > /dev/ttyACM0

echo -e '\e[1;8fAhoj ' >/dev/ttyACM0
```

7 Záver

V úvode práce sme sa zoznámili s fungovaním textových terminálov, s ich štandardmi a princípom komunikácie. Popísal som komunikáciu na sériových rozhraniach RS-232, RS-485, RS422, kompletný popis a komunikáciu na rozhraní USB, popis adresácie a inicializácie koncových bodov na tomto rozhraní, tak ako aj popis konfigurácie vnútorných descriptorov koncových bodov. S dosiahnutých informácií som bol schopný porovnať, vyhľadať a popísať potrebný procesor, s nutnosťou umožňovať podporu USB, interného oscilátoru, režim master, DIL prevedenie, pre zostavenie textového terminálu. Vybral som si vhodný LCD displej, popísal princíp fungovania, inicializácie a samotný zápis na displej.

Navrhol som adekvátne využitie portov procesoru na pripojenie dátovej zbernice LCD, ovládanie LCD, pripojenie MCU prostredníctvom ISP k programátoru. Realizoval som schémový návrh zariadenia, zdroja s čo najmenším počtom použitých súčiastok.

Vytvoril som programové vybavenie procesora a implementoval som ho. Pri implementácií navrhnutého softwaru, som testoval chovanie zariadenia - jeho reakciu na prijaté dáta, naplňovanie vnútorných bufferov, nezávislosť na jednotlivých riadiacích sekvenciách a programové vybavenie procesora som odladoval.
Pri návrhu programového rozhrania pre inicializáciu modulov, som daný modul niekoľko krát testoval a kód som dolaďoval pre čo najlepši výsledok oživenia modulou.
Vo výsledku, ako celku, som bol schopný zostaviť kompletný textový terminál.

Počas testovania pod operačným systémom Debian, som nepostrehol žiadne problémy v chováníi MCU, prípadne problémy (bugy) v kóde. Pri testovaní som do terminálu odosielal niekoľko hodín dáta a ku žiadnému problému, či chybe nedošlo.

Ďalší vývoj terminálu by sa mohol odvíjať pridaním farebného LCD modulu a tým doplniť kompletný VT štandard, prípadne doplniť terminál o USB klávesnicu a slúžil by ako kompletná, diaľkovo-ovládacia jednotka.

8 Zoznam príloh na CD

1. Spracovaný text bakalárskej práce vo formáte .pdf
2. Dátové listy a zdroje vo formáte .pdf
3. Zdrojové kódy pre mikropočítač vo vývojovom prostredí MPLAB
4. Skompilovaný súbor .hex pre programátor
5. Krátky návod README.txt na použitie terminálu

9 Reference

[1] *USB Class Codes* [web]. [cit. 7. 12. 2011] *Dostupný na* http://www.usb.org/developers/defined_class

[2] *CDC Class-Specific SubClasses* [web]. [cit. 16. 3. 2014] *Dostupný na* http://www.usblyzer.com/usb-communication-device-class-cdc-decoder.htms

[3] *CDC Class-Specific SubClasses* [web]. [cit. 16. 3. 2014] *Dostupný na* http://ww1.microchip.com/downloads/en/AppNotes/01247a.pdf

[4] *USB Framework for PIC18/PIC24/ PIC32* [online]. [cit. 2010-27-5] *Dostupný na* http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2680&dDocName=en537044

[5] Microchip. *AN1142: USB Mass Storage Class on an Embedded Host* [online]. [cit. 2. 5. 2008] *Dostupný na* http://ww1.microchip.com/downloads/en/AppNotes/01142A.pdf

[6] Microchip. *MPLAB IDE* [web]. [cit. 21. 2. 2014] *Dostupný na* http://www.microchip.com/pagehandler/en-us/family/mplabx/

[7] Microchip. *C30 C compiller* [web]. [cit. 10. 1. 2014] *Dostupný na* http://ww1.microchip.com/downloads/en/DeviceDoc/MPLABCdsPICv3_20_B.exe

[8] Microchip. *PIC24FJ64GB004 datasheet* [web]. *Dostupný na* http://ww1.microchip.com/downloads/en/DeviceDoc/39940d.pdf

[9] *Presto Programátor*[web]. [cit. 25. 2. 2013] *Dostupný na* http://www.asix.cz/prg_presto.htm

[10] *LCD - PVC160203PGL01* [web]. [cit. 15. 6. 2000] *Dostupný na* http://www.digchip.com/datasheets/parts/datasheet/000/PVC160203PGL01-pdf.php

[11] *Proteus Isic* [web]. [cit. 11. 11. 2008] *Dostupný na* http://proteus.soft112.com

[12] *VT100 Escape sequences* [web]. [cit. 2. 5. 2009] *Dostupný na* http://ascii-table.com/ansi-escape-sequences-vt-100.php